# ASSIGNMENT-6.3

NAME:Akhila Cheti

ROLL NUMBER:2303A51234

BATCH:18

-----------------------------------------------------------------------------------------------------

## Task Description #1: Classes (Student Class)

**PROMPT:**

# Create a Python class named Student.

# Add attributes: name, roll_number, branch.

# Use __init__ to initialize attributes.

# Add display_details() to print student info.

# Create one Student object.

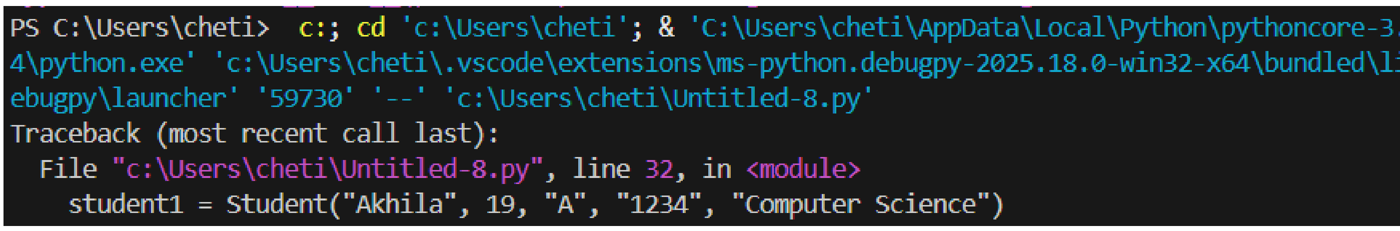
# Call the display_details() method.

**CODE:**

```
#Create a Python class named Student.
class Student:
    def __init__(self, name, age, grade):
        self.name = name
        self.age = age
        self.grade = grade

    def get_info(self):
        return f"Name: {self.name}, Age: {self.age}, Grade: {self.grade}"

#Add attributes: name, roll_number, branch.
```

```
        self.roll_number = roll_number
        self.branch = branch


#Use __init__ to initialize attributes.
        self.roll_number = roll_number
        self.branch = branch
#Add display_details() to print student info.
    def display_details(self):
        print(f"Name: {self.name}")
        print(f"Age: {self.age}")
        print(f"Grade: {self.grade}")
        print(f"Roll Number: {self.roll_number}")
        print(f"Branch: {self.branch}")


#Create one Student object.
student1 = Student("Akhila", 19, "A", "1234", "Computer Science")


#Call the display_details() method.
student1.display_details()
```

**OUTPUT:**



```
PS C:\Users\cheti>  c:; cd 'c:\Users\cheti'; & 'C:\Users\cheti\AppData\Local\Python\pythoncore-3
4\python.exe' 'c:\Users\cheti\.vscode\extensions\ms-python.debugpy-2025.18.0-win32-x64\bundled\l
ebugpy\launcher' '59730' '--' 'c:\Users\cheti\Untitled-8.py'
Traceback (most recent call last):
  File "c:\Users\cheti\Untitled-8.py", line 32, in <module>
    student1 = Student("Akhila", 19, "A", "1234", "Computer Science")
```

**EXPLANATION:**

This program uses Object-Oriented Programming to model student details. A Student class is created with attributes initialized using a constructor. A method is used to display student information. This improves code organization and reusability.

## Task Description #2: Loops (Multiples of a Number)

**PROMPT:**

#Write a Python function that prints the first 10 multiples of a given number using a for loop.

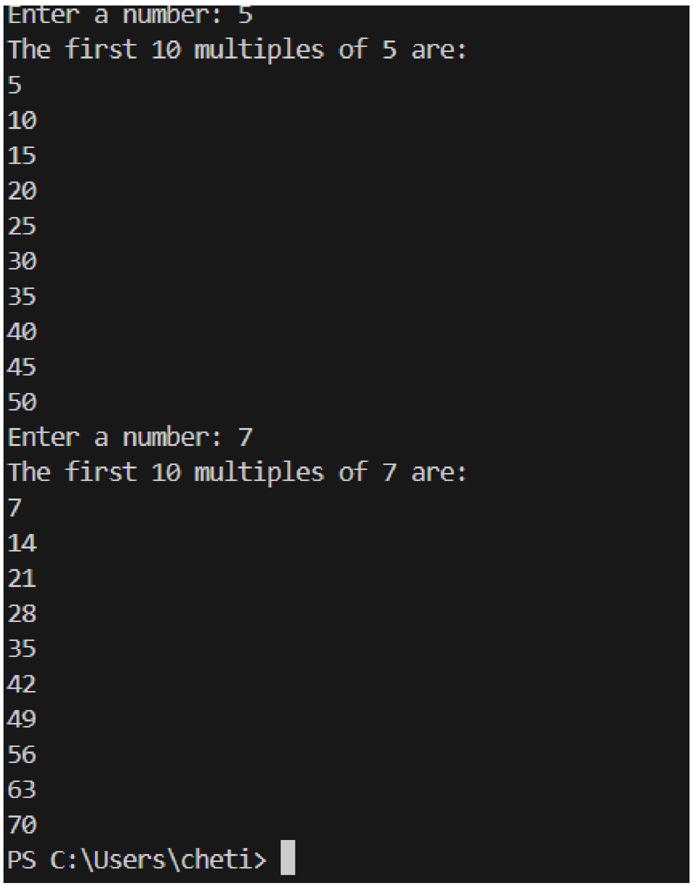
#The number should be taken as user input.

#Write the same program to print the first 10 multiples of a given number,use a while loop

**CODE:**

```
#Write a Python function that prints the first 10 multiples of a given number using a for loop.

#The number should be taken as user input.

def print_multiples():

    try:

        number = int(input("Enter a number: "))

        print(f"The first 10 multiples of {number} are:")

        for i in range(1, 11):

            print(number * i)

    except ValueError:

        print("Please enter a valid integer.")

print_multiples()


#Write the same program to print the first 10 multiples of a given number,use a while loop

def print_multiples_while():

    try:

        number = int(input("Enter a number: "))

        print(f"The first 10 multiples of {number} are:")

        count = 1

        while count <= 10:
            print(number * count)

            count += 1

    except ValueError:

        print("Please enter a valid integer.")

print_multiples_while()
```

**OUTPUT:**



```
Enter a number: 5
The first 10 multiples of 5 are:
5
10
15
20
25
30
35
40
45
50
Enter a number: 7
The first 10 multiples of 7 are:
7
14
21
28
35
42
49
56
63
70
PS C:\Users\cheti>
```

**EXPLANATION:**

- Uses for loop to generate first 10 multiples.
- Uses while loop as an alternative approach.
- Accepts user input for flexibility.
- Includes error handling for invalid input.

## Task Description #3: Conditional Statements (Age Classification)

**PROMPT:**

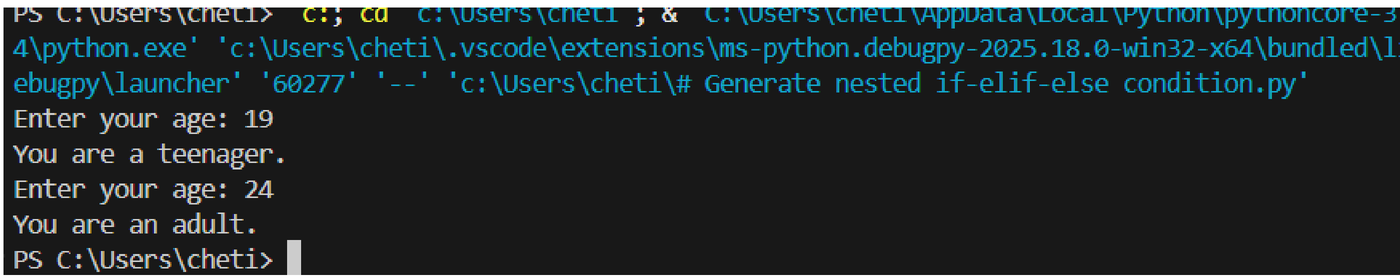
# Generate nested if-elif-else conditional statements to classify age groups(e.g., child, teenager, adult, senior).

# Generate the same classification using alternative conditional structures (e.g.,simplified conditions or dictionary-based logic).

## CODE:

```
# Generate nested if-elif-else conditional statements to classify age groups(e.g., child, teenager, adult, senior).

age = int(input("Enter your age: "))

if age < 0:

    print("Invalid age.")

elif age < 13:

    print("You are a child.")

elif age < 20:

    print("You are a teenager.")

elif age < 65:

    print("You are an adult.")

else:

    print("You are a senior citizen.")


# Generate the same classification using alternative conditional structures (e.g.,simplified conditions or dictionary-based logic).

def classify_age(age):

    """Classify age groups using dictionary-based logic."""

    age_groups = {
        range(0, 13): "You are a child.",
        range(13, 20): "You are a teenager.",
        range(20, 65): "You are an adult.",
        range(65, 150): "You are a senior citizen."
    }
    for age_range, message in age_groups.items():
        if age in age_range:
            return message
    return "Invalid age."
user_age = int(input("Enter your age: "))
print(classify_age(user_age))
```

**OUTPUT:**



```
PS C:\Users\cheti>  c:; cd  c:\Users\cheti ; &  C:\Users\cheti\AppData\Local\Python\pythoncore-3
4\python.exe' 'c:\Users\cheti\.vscode\extensions\ms-python.debugpy-2025.18.0-win32-x64\bundled\l
ebugpy\launcher' '60277' '--' 'c:\Users\cheti\# Generate nested if-elif-else condition.py'
Enter your age: 19
You are a teenager.
Enter your age: 24
You are an adult.
PS C:\Users\cheti>
```

**EXPLANATION:**

- Applies conditional statements for decision making.
- Classifies age into child, teenager, adult, and senior.
- Demonstrates both nested and dictionary-based logic.
- Improves readability and logical structure.

## Task Description #4: For and While Loops (Sum of First n Numbers)

**PROMPT:**

# Generate a sum_to_n() function using a for loop.

# Generate an alternative implementation using a while loop or a mathematical formula.

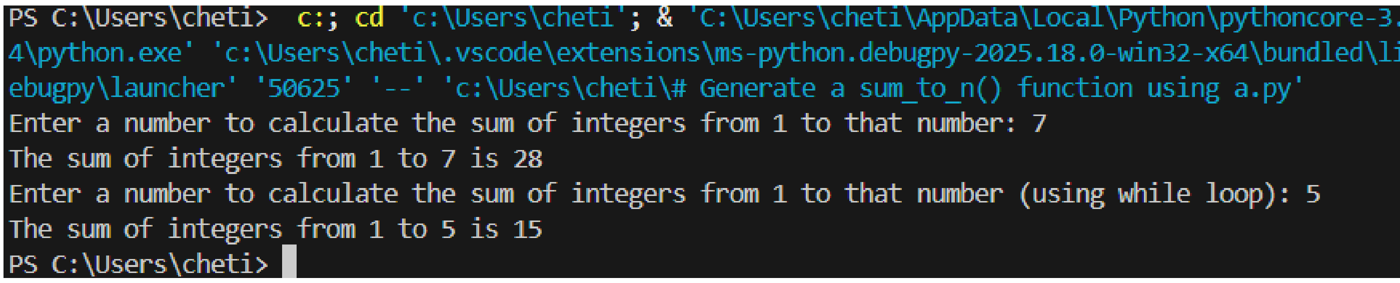
**CODE:**

```
# Generate a sum_to_n() function using a for loop.
def sum_to_n(n):
    """Calculate the sum of integers from 1 to n."""
    total = 0
    for i in range(1, n + 1):
        total += i
    return total
# Example usage:
number = int(input("Enter a number to calculate the sum of integers from 1 to that number: "))
print(f"The sum of integers from 1 to {number} is {sum_to_n(number)}")
# Generate an alternative implementation using a while loop or a mathematical formula.
def sum_to_n_while(n):
    """Calculate the sum of integers from 1 to n using a while loop."""
    total = 0
    i = 1
    while i <= n:
        total += i
        i += 1
    return total
# Example usage:
```

number = int(input("Enter a number to calculate the sum of integers from 1 to that number (using while loop): "))

print(f"The sum of integers from 1 to {number} is {sum_to_n_while(number)}")

**OUTPUT:**



```
PS C:\Users\cheti>  c:; cd 'c:\Users\cheti'; & 'C:\Users\cheti\AppData\Local\Python\pythoncore-3.
4\python.exe' 'c:\Users\cheti\.vscode\extensions\ms-python.debugpy-2025.18.0-win32-x64\bundled\li
ebugpy\launcher' '50625' '--' 'c:\Users\cheti\# Generate a sum_to_n() function using a.py'
Enter a number to calculate the sum of integers from 1 to that number: 7
The sum of integers from 1 to 7 is 28
Enter a number to calculate the sum of integers from 1 to that number (using while loop): 5
The sum of integers from 1 to 5 is 15
PS C:\Users\cheti>
```

**EXPLANATION:**

- Calculates sum using a for loop.
- Provides alternative implementation using a while loop.
- Uses functions for reusability.
- Demonstrates loop control and arithmetic operations.

## Task Description #5: Classes (Bank Account Class)

**PROMPT:**

#Write a Python BankAccount class with deposit(), withdraw(), and check_balance() methods

# Create an instance of BankAccount

# Demonstrate the functionality of each method

**CODE:**

```
#Write a Python BankAccount class with deposit(), withdraw(), and check_balance() methods

class BankAccount:

  def __init__(self, initial_balance=0):

    self.balance = initial_balance


  def deposit(self, amount):
    if amount > 0:
        self.balance += amount
        print(f"Deposited: ${amount}")
    else:
        print("Deposit amount must be positive.")

  def withdraw(self, amount):
    if 0 < amount <= self.balance:
        self.balance -= amount
        print(f"Withdrew: ${amount}")
    else:
        print("Insufficient funds or invalid withdrawal amount.")

  def check_balance(self):
    print(f"Current balance: ${self.balance}")
# Create an instance of BankAccount
account = BankAccount(100)  # Initial balance of $100
account.check_balance()
account.deposit(50)
account.check_balance()
account.withdraw(30)
account.check_balance()
account.withdraw(150)  # Attempt to withdraw more than the balance
account.check_balance()
# Demonstrate the functionality of each method
account.deposit(-20)  # Attempt to deposit a negative amount
account.check_balance()
account.withdraw(50)
```

account.check_balance()

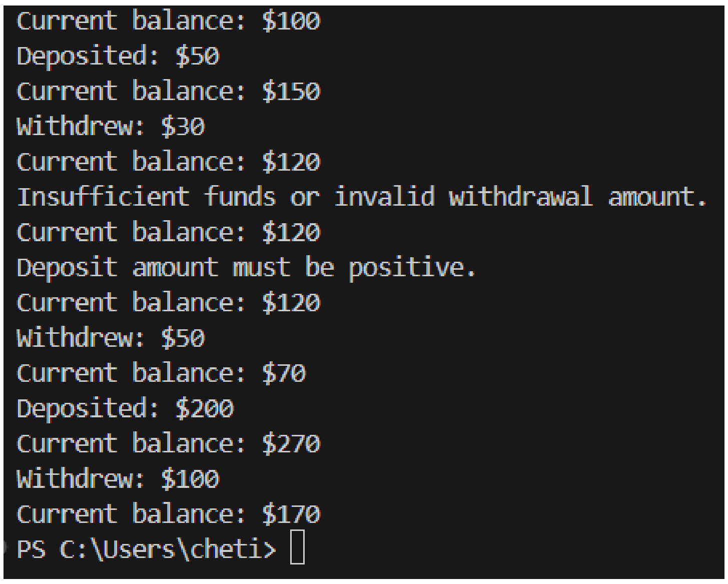
account.deposit(200)

account.check_balance()

account.withdraw(100)

account.check_balance()

**OUTPUT:**



```
Current balance: $100
Deposited: $50
Current balance: $150
Withdrew: $30
Current balance: $120
Insufficient funds or invalid withdrawal amount.
Current balance: $120
Deposit amount must be positive.
Current balance: $120
Withdrew: $50
Current balance: $70
Deposited: $200
Current balance: $270
Withdrew: $100
Current balance: $170
PS C:\Users\cheti>
```

**EXPLANATION:**

- Models real-world banking system using a class.
- Implements deposit, withdraw, and balance check methods.
- Ensures validation for transactions.
- Demonstrates encapsulation and data security.